並列DDFアルゴリズム(数式処理における理論とその応用の研究)

AUTHOR(S):

藤瀬, 哲朗; 村尾, 裕一

数理解析研究所講究録
941 巻 1996 年 99-112

13.

並列 DDF アルゴリズム

藤瀬 哲朗 (三菱総合研究所)
村尾 裕一 (東京大学大型計算機センター)

13.1 はじめに

極めて高次の 1 変数多項式の因数分解において，DDF(Distinct Degree Factorization: 次数別因数分解) は，既約性の判定や問題の break-down (より低い次数の多項式の因数分解) のために有用である．DDF のアルゴリズムとして，近年，von zur Gathen & Shoup のアルゴリズム [9] と Kaltofen & Shoup のアルゴリズム [14] が開発された．これらのアルゴリズムは，大雑把な分解 (**coarse DDF** ステップ) と最終的に必要な分解 (**fine DDF** ステップ) の 2 段階からなり，また，complexity に関しては次の性質を持つ (但し，次数を n とする)．

- いずれも所要スペースは $O(n^{3/2})$.
- 計算量が $O(n^{\alpha})$ (α は 2 前後) と高速である．

一般に，対象とする多項式の次数 n が $\gg 1$ の時，多項式基本演算に漸近的高速アルゴリズムを用いることは，いかなる因数分解アルゴリズムにおいても効率上有効である．上述の DDF アルゴリズムの (理論的) 高速性も，幾種類かの多項式演算における高速アルゴリズムの利用に依るところが大きい．換言すれば，$n \gg 1$ の場合に如何なるアルゴリズムにおいても必要となる，多項式演算の高速アルゴリズムが効率良く実現されさえすれば，上述の DDF アルゴリズムは，他のアルゴリズムとの比較において，単に計算量における漸近的な振舞いに優れるだけでなく，高い実用性を有するものと思われる．本稿では，この新しい DDF アルゴリズムの並列化の方法について考察する．

■動機とアイディア

素数 p が一語長に収まる程度の大きさの場合，$\mathbf{Z}_p$ 係数の多項式の因数分解にはベクトル処理が有効であることは，筆者らが [16] や [17] で実証した．しかし，そこで実現したアルゴリズムは，Berlekamp の行列 Q $(n \times n)$ を用いるため，$O(n^2)$ のスペースが必要となり，数 GB のメモリを実装しているような最大級のベクトル計算機をもってしても，10^4 次程度が限界である (また，$GF(p^k)$ 係数の場合や p の大きさが一語長を越える場合には直接的なベクトル処理が不可能であり，このため，[8] の問題へのチャレンジにも適用できない)．他方，超並列計算機においては，数百 GB のメモリを分散実装するシステムも登場してきており，並列 block Wiedemann アルゴリズム [12] を用いて，行列 Q の反復法による消去を行えば [13]，さらに高い次数 (10^5程度) の多項式の因数分解も可能と思われる．このような状況において，我々は，ベクトル処理で用いた旧来の因数分解アルゴリズムにかわる，より良いアルゴリズムの探求を始め [11]，その結果，冒頭に挙げたアルゴリズムが $O(n^{3/2})$-space であることに着目し，その実用化を検討し始めたのが本研究のそもそものきっかけである．

これら最新のアルゴリズムの並列化を検討するに到った，より積極的な動機は極めて単純である．

- ▷ DDF アルゴリズムにおける GCD 計算は，基本的に探索問題である．よって，[6] や [16] に指摘したように，並列処理による効率化が期待される．
- ▷ fine DDF ステップ中には，一つの場合を除き，代数的独立性を容易に見い出すことができ，並列処理が可能である．

より興味深いのは，並列化においては，計算ステップの適切なスケジューリングを予め行い，全体の計算時間の短縮を図ることが本質的であるが，第 4 回日本数式処理学会大会において指摘したように [17]，並列処理計算量に関し，次の事柄が予想されることである．

- ▷ 各 fine DDF ステップは，coarse DDF の逐次処理と同時実行することにより，計算量的には隠蔽されるであろう．
- ▷ 代数的独立性のない一つの fine DDF ステップについては，同程度の計算量になると予想される coarse DDF の逐次処理全体により隠蔽可能であろう．
- ▷ さらに，この fine DDF ステップも並列化するのであれば，coarse DDF ステップから recursion を行えばよく，この場合にも，その計算量は他の fine/coarse DDF ステップにより隠蔽されるであろう．
- ▷ 並列度は $O(n^{1/2})$ 程度，即ち，$n^{1/2}$ の数倍の台数のプロセッサを利用すれば，上記の隠蔽を完全に行うことができるであろう．

我々の興味の焦点は，これらの技巧の正当性を明らかにし，それらを駆使した並列処理アルゴリズムを開発することである．上述の技巧とアイディアは，GCD 計算の繰り返しにより因子の分離を行うステップだけに関連するものだが，本稿では，その前段階で行う，そのために必要な式の計算も含め，Kaltofen & Shoup アルゴリズムの全体を並列化し，その (計算量的) 検討を行う．特に，本稿で示す

アルゴリズムでは，上記の技巧をさらに進め，

- 代数的独立性のない fine DDF ステップに相当する GCD 計算を，より早期のステップと並行処理 (計算量的に隠蔽) することにより，find/coarse DDF ステップの簡潔な並列化を可能とする．

基本となるアルゴリズムのより詳細については，参考文献に挙げる原論文や [17] を参照されたい．

13.2 DDF アルゴリズム — おさらい

先ず，基本式に関する表記法を導入する．

$\boxed{\mathbf{F}_q}$ ··· 有限体 (要素数は q)．

$\boxed{F(x)}$ ··· $\mathbf{F}_q$ 上の無平方 (square-free) で monic な一変数多項式とし，その次数を $\boxed{n} = \deg(F)$ とおく．

$\boxed{f_k(x)}$ ··· $F(x)$ の因子のうち次数が k である全ての既約因子の積とする．

Distinct Degree Factorization (以下 DDF) とは，$F(x)$ が与えられた時，この因子多項式 $f_k(x)$ の全てを求めることである．

13.2.1 従来の DDF アルゴリズム

良く知られた基本的事実を記す．

事実 1 ：任意の多項式 $P(x) \in \mathbf{F}_q[x]$ について，$P(x)^q = P(x^q)$．

事実 2 ：$x^{q^k} - x$ は，次数が k を割り切るような全ての monic な既約多項式 $\in \mathbf{F}_q[x]$ の積である．

従来から良く知られた DDF アルゴリズムとして，事実 2 に基づき，$k = 1, 2, \ldots$ と昇順に $f_k(x)$ を分離していくという方法がある．その具体的な記述を以下に示す．

$k \Leftarrow 1$; $W \Leftarrow x$;
while $2k \leq \deg(F(x))$ **do**
  $W \Leftarrow W^q \bmod F(x)$; (* $x^{q^k} \bmod F(x) = \lambda_k(x)$ *)
  $f_k(x) \Leftarrow \gcd(F(x), W - x)$;
  if $f_k(x) \neq 1$ **then** { $F(x) \Leftarrow F(x)/f_k(x)$; $W \Leftarrow W \bmod F(x)$; }
  $k \Leftarrow k + 1$;

このアルゴリズムは，基本的には，因子の探索を k に関する **exhaustive search** により行う方法で

あることに注意.

■並列化するならば ...

上のアルゴリズムが単純な探索問題であったとしたら，並列化は $k=1,2,...$ に関する **while** ループを適宜分割するのが最も簡単であろう．しかし，実際には，次の 2 つの問題が存在するために，この単純な方法を用いることはできない.

(1) $x^{q^k} \bmod F(x)$ の計算における，k についての逐次性.

(2) $\hat{f}_k \Leftarrow \gcd(F(x), x^{q^k}-x)$ は，$\prod_{i \mid k} f_i(x)$ に等しく，f_k 以外の因子を含む.

(2) の問題，即ち，$\hat{f}_k$ から f_k を求める方法について，若干の考察を行う．$\hat{f}_k$ 中の余分な因子 $\hat{f}_k/f_k$ は，k の全ての因子 j に対する f_j からなるが，各 f_j は，エラトステネスのふるい型のループとネットワークのようなものがあれば，効率良くシステマティックに得られるかもしれない．より良い方法は，k の全ての素因子 j に対し，$\hat{f}_{k/j}$ との共通因子を順次取り除いていく方法であろう．例えば，$\hat{f}_{12} = f_1 f_2 f_3 f_4 f_6 f_{12}$ だが，

- $k=12$ の素数因子 3 に対する $\hat{f}_4 = f_1 f_2 f_4$ により，$W \Leftarrow \hat{f}_{12}/\gcd(\hat{f}_{12}, \hat{f}_4) = f_3 f_6 f_{12}$ を得，
- 素数因子 2 に対する $\hat{f}_6 = f_1 f_2 f_3 f_6$ により，$W/\gcd(W, \hat{f}_6) = f_{12}$ を得る.

但し，上記のいずれにおいても，k の素因数分解や，k に関する因子間の強い依存関係とその事により生ずる多大な通信量が問題となり，効率良く並列化するのは容易ではない.

一方，探索問題においては，先ず探索の範囲を大雑把に分割し，次第にその範囲を狭めていくという方法が，最も簡単で一般的な効率化の方法である．次節に示す新しいアルゴリズムでは，この方法を用いており，並列化にも適合する.

13.2.2 最近/最新の DDF アルゴリズム

DDF の手続きを探索問題と見做せば，exhaustive search を行う旧来のアルゴリズムは，特に既約因子の次数の分布に偏りがある場合，効率が良くない．1992 年に発表された von zur Gathen & Shoup のアルゴリズム，及び，さらにそれを改良した Kaltofen & Shoup のアルゴリズムは，次のようにして，この問題を解決する.

(1) 先ず，ある区間 $(\ (k-1)l, kl\ ]$ 内の次数を持つ既約因子の積 $\boxed{F_k(x)}$ へと (次数の幅 l を持たせて大雑把に) 分解する．$\boxed{l}$ としては，$\lfloor\sqrt{n}\rfloor$ または $\lceil\sqrt{n/2}\rceil$ を用い，$\boxed{m} = \lfloor n/(2l)\rfloor$ とおく.

(2) 各 $F_k(x)$ を $f_i(x), (k-1)l < i \leq kl$ に分解する.

前者を **coarse DDF**，後者を **fine DDF** と呼ぶ [14].

事実 3 ：以降，$\boxed{\lambda_i(x)} \stackrel{\text{def}}{=} x^{q^i} \bmod F(x)$ と書く．明らかに次の等式が成り立つ.

$$\lambda_i(\lambda_j) \bmod F = \lambda_{i+j} \text{ 及び } {\lambda_i}^q \bmod F = \lambda_{i+1}.$$

■ von zur Gathen & Shoup の DDF アルゴリズム [9]

以下に概略を示す.

(GS0) $\lambda_i(x), 0 \le i \le l-1$ 及び$\lambda_{lk+1}(x), 0 \le k \le m$ を計算しておく.
$G(Y,x) \Leftarrow \prod_{i=0}^{l-1} (\lambda_i(Y) - x) \bmod F(Y)$ を計算しておく.

(GS1) $0 \le k \le m$ なる全ての k に対し $G(\lambda_{lk+1}(x), x) = \prod_{i=0}^{l-1} (x^{q^{lk+1+i}} - x) \bmod F(x)$ を求めた後, $k = 0, 1, ..., m$ の昇順で $G(\lambda_{lk+1}(x), x)$ との gcd 計算を繰り返し, $F_{k+1}(x)$ を得る. ..... **(coarse)**

(GS2) 各 $F_{k+1}(x)$ に対し: $0 \le i \le l-1$ なる全ての i に対し$\lambda_{lk+1}(\lambda_i(x)) \bmod F_{k+1}(x)$ を求めた後, $i = 0, 1, ..., l-1$ の昇順に $\gcd(\lambda_{lk+1}(\lambda_i(x)) - x, F_{k+1}(x))$ を計算し, $f_i(x)$ を得る. ..... **(fine)**

■ Kaltofen & Shoup の DDF アルゴリズム [14]

次の事実を用いることにより, (GS2) における $\lambda_{lk+1}(\lambda_i(x)) \bmod F_{k+1}(x)$ の計算を省くという改良を施し, 同時に, 計算量的技巧をさらに進めたアルゴリズムである.

事実 4 : $i > j \ge 0$ なる任意の整数 i 及び j に対し, 多項式 $x^{q^i} - x^{q^j} = \left(x^{q^{i-j}} - x\right)^{q^j} \in \mathbf{F}_q[x]$ は, $(i-j)$ の因子を次数とする任意の既約多項式 $\in \mathbf{F}_q[x]$ で割り切れる.

以下にその概略を示し, 図 1 にアルゴリズムの具体的手続きを記述する.

(KS0) $\lambda_i(x), 0 \le i \le l-1$ 及び$\lambda_{lk}(x), 1 \le k \le m$ を計算しておく.
$K_l(Y,x) \Leftarrow \prod_{i=0}^{l-1} (Y - \lambda_i(x)) \bmod F(x) = (-1)^l G(x,Y)$ を計算しておく.

(KS1) $0 \le k \le m$ なる全ての k に対し $K_l(\lambda_{lk}(x), x) = \prod_{i=0}^{l-1} \left(x^{q^{lk}} - x^{q^i}\right) \bmod F(x)$ を求めた後, $k = 1, 2, ..., m$ の昇順で $K_l(\lambda_{lk}(x), x)$ との gcd 計算を繰り返し, $F_k(x)$ を得る. ..... **(coarse)**

(KS2) 各 $F_{k+1}(x)$ に対し, $i = l-1, l-2, ..., 0$ の降順に $\gcd(\lambda_{lk}(x) - \lambda_i(x), F_k(x))$ を計算し, $f_i(x)$ を得る. ..... **(fine)**

どちらのアルゴリズムにおいても, k 及び i に関する繰り返しは上記の順に行うこと, 及び, fine DDF のステップ (GS2) 及び (KS2) は exhaustive search であることに注意. 表1 に, 両アルゴリズムにおける各ステップ毎の計算量をまとめる. 表中, $M(n)$ は n 次多項式の乗算の計算量を, $MC(n,N)$ は n 次多項式による N 個の modular composition (後述) の計算量を表す.

(1) $\lambda_0(x) \Leftarrow x$; $\lambda_1(x) \Leftarrow x^q \bmod F(x)$;
for $i = 0$ **to** $\lceil \log_2 l \rceil - 1$ **do** { $\lambda_{k+2^i}(x) \Leftarrow \lambda_{2^i}(\lambda_k(x))$, $1 \le k \le \max(2^i, l - 2^i)$; }
(* $\lambda_k(x)$ for $k = 2$; $3 \sim 4$; $5 \sim 8$; $9 \sim 16$; $17 \sim 32$; ... *)

(2) **for** $i = 0$ **to** $\lceil \log_2 m \rceil - 1$ **do** { $\lambda_{(k+2^i)l}(x) \Leftarrow \lambda_{2^i l}(\lambda_{kl}(x))$, $1 \le k \le \max(2^i, m - 2^i)$; }
(* $\lambda_{kl}(x)$ for $k = 2$; $3 \sim 4$; $5 \sim 8$; $9 \sim 16$; $17 \sim 32$; ... *)

(3) $K_l(Y, x) \Leftarrow \prod_{i=0}^{l-1} (Y - \lambda_i(x)) \bmod F(x)$;

(4) $\Delta_k \Leftarrow K_l(\lambda_{kl}(x), x) \bmod F(x)$ for $1 \le k \le m$;

(5) $k \Leftarrow 1$; **while** $2k \le \deg F$ **do** { $\gcd(\Delta_k, F) \Rightarrow F_k$; $F \Leftarrow F/F_k$; $k \Leftarrow k + 1$; }
(* coarse DDF: ($(k-1)l$, kl] 次の F の既約因子の分離 *)

(6) for $1 \le k \le m$, do the following (* fine DDF *)
$i \Leftarrow l - 1$; $\bar{\lambda}_{kl} \Leftarrow \lambda_{kl} \bmod F_k$;
while $2(kl - i) \le \deg F_k$ **do** { $\gcd(\bar{\lambda}_{kl} - \lambda_i, F_k) \Rightarrow f_{kl-i}$; $F_k \Leftarrow F_k / f_{kl-i}$; $i \Leftarrow i - 1$; }

図 1 Kaltofen & Shoup アルゴリズムの逐次処理手続き

表 1 計算量のまとめ

step	calculation	von zur Gathen & Shoup	Kaltofen & Shoup	space
(1)	λ_1	$O(M(n)\log q)$		n
	$\lambda_2, ..., \lambda_l$	$O(lM(n)\log q)$	or $O(\sum_{i=1}^{\lceil \log_2 l \rceil - 1} MC(n, 2^{i-1}))$	ln
(2)	$\lambda_{2l}, ..., \lambda_{ml}$	$O(mMC(n,1))$	or $O(\sum_{k=1}^{\lceil \log_2 m \rceil - 1} MC(n, 2^{k-1}))$	mn
(3)	$G(Y,x)$ or $K_l(Y,x)$	$O(M(ln)\log l)$		ln
(4)	$G(\lambda_{kl}, x)$ or $K(\lambda_{kl}, x)$, $0 \le k \le m$	$O(mlM(n))$	or $O(M(n)M(m)\log m)$	mn
(5)	coarse DDF	$O(mM(n)\log n)$		
(6)	$\lambda_i \bmod F_k$	$O(lM(n))$		ln
	$\lambda_{kl} \bmod F_{k+1}$ or $\lambda_{kl+k} \bmod F_k$	$O(mM(n))$		n
	fine DDF	$O(lM(n)\log n)$		

13.3 並列化の方針

本節では，本稿の主題である Kaltofen & Shoup のアルゴリズムの並列化について，その方針を列挙する．

- 前節のアルゴリズム記述中の，$1 \leq k \leq \ldots$ や$\prod_i$ の部分及び (5) と (6) のループといった，粒度の荒い部分を並列化の対象として考える．
- ステップ (1)~(4) は，主として，計算量的な技法を如何にうまく並列化するかだけが問題である．
- 並列化の焦点は，冒頭に記したとおり，(5) と (6) における探索を融合し，(ループの最後まで見つからないという最悪の場合を) 高速化することである．
  - ステップ (5) の **while** ループは，§ 13.2.1 の場合と同様，逐次実行する必要がある．
  - 異なる k に対する fine DDF は代数的に独立である．よって，ステップ (6) の for $1 \leq k \leq m$ の k は任意の順序で実行することができ，また，ステップ (5) で F_k が得られ次第パイプライン的に，並列に実行可能である．
  - $k > 1$ の時，$2((k-1)l+1) > kl$ ゆえ，一つの F_k 内の fine DDF は (i について) 代数的に独立であり，並列実行可能である．
  - F_1 の fine DDF を並列化するには，再帰的に計算するのもよい．
  - 再帰した場合 ($l \to l^{1/2}$) に必要となる全てのλ_i は，$l^{1/2} \times l^{1/2} = l$ ゆえ，ステップ (1) で計算済みである．
  - その場合でも，新たな $F = F_1$ を用いて，mod$F$ の簡約化をしておくことは有用であろう (普通の計算では，全ての λ_i が必要になることは稀だろうから，どの時点でやるかは難しい．以下同様)．
  - $K_l(Y,x)$ についても，計算量の議論における一般的手法にに従い l が 2 の (2 の) 冪乗と仮定し，後述のとおり binary-tree multiplication により計算したとすれば，$K_{\sqrt{l}}(Y,x)$ も計算済みである．一方，$K_{l/2}(Y,x)$ も計算済みであるから，再帰後は，$\sqrt{l}$ 毎の coarse DDF の代わりに，$l/2$ 毎の divide-and-conquer が良い ($\Leftarrow$ ステップ (5) のループの逐次性)．
- ステップ (2) とステップ (3) は独立な計算 (同時実行可能) である．
- ここで，$K_l(Y,x)$ の計算が，ステップ (1) における i の小さい値に対する λ_i の計算が完了し次第パイプライン的に開始できることに気付けば，同様に，ステップ (1) の **for** ループと並行して $\gcd(\lambda_i - x, F) \Rightarrow f_i$ の計算を別のプロセッサで行えばよいことに気付く．これにより，**ステップ (5) における F_1 の分離及びステップ (6) における F_1 の fine DDF は不要となり，上記の recursion も行わなくて済む**．
- 我々の経験 ([15] の計算について，SparcCenter で共有メモリを直接使った場合と SparcStation10 × 8 台を PVM を介して使った場合の比較) から，通信コストは無視する．

13.4 道具と計算量

並列アルゴリズムを考える場合，プロセッサができる限り活用されるよう，各計算ステップを旨くスケジューリングすることが肝要である．計算量の解析は，そのために最低限必要な測度となる．

13.4.1 基本演算

$u, v \in R[z]$ (R は $\mathbf{F}_q$ 又は $\mathbf{F}_q[x]/(F)$) とし，これらの次数 n は $\gg 1$ とする．多項式の基本演算について，漸近的高速アルゴリズムの計算量等の要点を以下にまとめる．計算量や space は，R 要素に対するものを単位とする．詳細は [1] や [2] や [18] を参照．

(多項式の乗算) $u \times v \in R[z]$ ..... $O(M(n)) \leq O(n^2)$

- $n \geq \deg u, \deg v \gg 1$ の場合，DFT(Discrete Fourier Transform) が有用である [19]．この時，$O(M(n)) = O(n \log n \log\log n)$ [5].
- $(q-1)$ の適当な除数 d で畳み込みを行うが，一般の n に対しては，実際の効率上 d の選択が重要となる．
- $d < \deg(uv)$ の時は，$z^{n/(2d)}$ を別変数と扱うようにして recursion を行う．
- 多変数の場合は，逆に，Kronecker のトリックを使うように行う．
- 変換結果のための $O(n)$-space が必要となる．

(多項式の剰余) $w \Leftarrow u \bmod v = u - Qv \in R[z]$ (但し v は monic) ..... $O(M(n))$

- 以降において，一般に，$z^{\deg f} f(1/z)$ を $\boxed{\widehat{f(z)}}$ と書く．
- $k = \deg Q = \deg u - \deg v$ とし，$V \Leftarrow \widehat{v} \bmod z^{k+1}$ とする．
- $\phi V \equiv 1 \bmod z^{k+1}$ なる $\phi \in R[z]$ を Newton 法:

$$\begin{aligned} \phi_1(z) &\Leftarrow 1, \\ \phi_{i+1}(z) &\Leftarrow \phi_i(z)(2 - V(z)\phi_i(z)) \bmod z^{2^i}, \quad i = 1, 2, ..., \lceil \log_2 k \rceil \end{aligned}$$

  で求めておけば，$\widehat{Q} \Leftarrow \widehat{u}\phi \bmod z^{k+1}$.
- 以降では，$v \leftarrow F$ で，$\deg u \leq 2(\deg F - 1)$ に対する計算が瀕出するが，F に対する ϕ は一度だけ計算しておけば良く，また，一般に $\bmod F$ 演算の計算量は $O(M(\deg F))$ である．

(多項式 GCD) $\gcd(u, v)$ の計算 ..... $O(M(n) \log n)$

尚，これらの基本演算においても様々な並列処理が可能と思われるが，$R = \mathbf{F}_q$ の場合には，粒度が極めて細かくなるので本稿では扱わない．

13.4.2 応用計算

ステップ (1) ~ (4) で必要となる演算をまとめる.

- multiple (複数の k に対する) modular composition: $\lambda_{(k+2^i)_s} \Leftarrow \lambda_{2^i s}(\lambda_{ks})$ or $\lambda_{ks}(\lambda_{2^i s})$
- binary-tree multiplication: $\prod_{i=1}^{2^{L+1}}(Y-\alpha_i) = \left(\prod_{i=1}^{2^L}(Y-\alpha_i)\right)\left(\prod_{i=1}^{2^L}(Y-\alpha_{i+2^L})\right)$.
  ▷ 但し, $K_{2l}(Y,x) = K_l(Y,x)K_l(Y,\lambda_l(x))$ にも注意.
- multi-point evaluation: $\Delta_k \Leftarrow K_l(\lambda_{kl}(x),x) \bmod F$ for $1 \le k \le m$, 他.
  ▷ Chinese remaindering (binary-tree multiplication を利用) を用いる.
  ▷ Chinese remaindering による剰余計算の各段階で必要となる先述の ϕ 多項式の全ても, binary-tree 的に構成でき, Newton iteration は不要となる (詳細は [7]).

これらに関する計算法と計算量をまとめる. 以降において, $M(n)$ 及び $MM(n)$ は, 各々 $\mathbf{F}_q$上の n 次多項式 及び $\mathbf{F}_q$要素の $n \times n$ 行列の積の計算量を表す. また, 計算量においては, log や poly-log 及びこれらの冪乗を省略したり $n^{o(1)}$ と表すことがある.

(modular composition) .....[9, FACT 5.1]

$(n-1)$ 次多項式 $P(x), Q(x) \in \mathbf{F}_q[x]/(F)$ に対し, $P(Q(x)) \bmod F$ を求める.

(mc-i) Horner 流 : $O(nM(n))$-time, $O(n)$-space.

(mc-ii) $F(x) = x^n$ の場合 [3] : $O\left(n^{1/2}MM(n^{1/2}) + n^{1/2}M(n)\right)$-time, $O\left(n^{3/2}\right)$-space.

(mc-iii) 一般の場合 ((ii) の一般化) : $O\left(n^{1/2}MM(\sqrt{nk})\log n + n^{3/2}M(t)t^{-1} + M(n)(\log n)^3\right)$-time, $O(n\log n)$-space. 但し $t = \lceil \log n/\log q \rceil$.

(multi-point evaluation) .....[1], [9]

多項式 $P(Y) \in R[Y]$ について, k個の値 $P(\alpha_i)$, $\alpha_i \in R$, $0 \le i < k$ を同時に求める. $P(Y) \bmod (Y-\alpha_i)$ の計算に等価. 但し, $O()$ は R 演算.

(me-i) $\deg P \le k$ の場合 : [1] の Chinese remaindering の方法 ($q_{j,s} \Leftarrow \prod_{i=0}^{2^s-1}(Y-\alpha_{j+i})$ を計算した後, $P(Y) \bmod q_{j,s}$ を s の大きい方から divide&conquer で計算していく) により, $O(M(k)\log k)$.

(me-ii) $\deg P \ge k$ の場合 : $d = \deg P$ として, 先ず $P(Y)$ を Y^k 毎に折り畳み, その後は (i) と同様. よって, $O\left(\left(\frac{d}{k} + \log k\right)M(k)\right)$ [9, LEMMA 2.2].

(multiple modular composition) .....[9], [14]

$(n-1)$ 次多項式 $P(x), r_1(x), r_2(x), ..., r_k(x) \in \mathbf{F}_q[x]/(F)$ に対し, $P(r_i(x)) \bmod F$ または $r_i(P(x)) \bmod F$ を, $1 \le i \le k$ に対し同時に求める. 但し, $n > k$.

(mmc-i) (me-ii) の利用 : $O((n/k + \log k)M(kn))$-time, $O(kn)$-space [9].

(mmc-ii) [14, **Lemma 3**] の方法 ([3] の multi 化). $t = \lceil \sqrt{nk} \rceil$ 項毎に折り返し, 係数行列を利用. $O((t + nk/t)M(n) + n/t\, MM(t))$-time, $O\left(nk + \sqrt{n^3 k}\right)$-space.

13.5 各計算における並列化の方法について

binary-tree 型の計算や doubling 型の計算 ... 以下に列挙する計算が該当する．同一レベルの計算を並列化する．

- 2 変数多項式 $K_l(Y,x)$ を計算する binary-tree multiplication
- Chinese remaindering における，2 変数多項式の binary-tree multiplication 及びそれに対する ϕ 多項式
- $\lambda_{(k+2^i)_s}$ の計算
- 多項式 g の冪乗計算: $\{g^2\} \to \{g^4, g^3\} \to \{g^8, ..., g^5\} \to \cdots$

multi-point evaluation

- multi-point の分を並列化し，各々では Horner を用いるのが最も簡単である．
- point 自体が多項式 ($\in \mathbf{F}_q[x]/(F)$ の場合，各 evaluation において，上の doubling による冪乗計算及びそれに続く係数との乗算を並列化することも考えられる．
- Chinese remaindering による場合は，上記のとおり．

multiple modular composition

- multi-point evaluation と見做せば，前項のとおり．
- (mmc-ii) の 計算量の内，$M(n)$ の分は冪乗計算と Horner ゆえ上記の方法が可能だし，また，dominant な行列乗算の分については，粒度との兼ね合いを考えて，適切な t の値の選択と行列の分割が必要である．

$\lambda_{(k+2^i)_s}$の計算 ... ステップ (1) 及び (2)

- 最も単純には k に関して並列化を行い，doubiling により必要な式の全てを計算する．doubling の回数 $O(\log_2 n^{1/2})$ だけの逐次性がある．
- 各レベルでは，$2^i \leq n^{1/2}$ の並列度で，計算時間の上限は，Horner 流で $O(nM(n))$，また，Brent&Kung 流で $O(n^{1/2}(M(n) + MM(n^{1/2})))$.

$K_l(Y,x)$ の計算 ... ステップ (3)

- binary-tree multiplication を用いる．逐次性と並列度は $O(\log_2 l)$ である．
- レベル k では，Y について 2^{k-1} 次 の多項式同志の独立な乗算が $l/2^k$ 個．よって計算時間は $O(M(2^{k-1}n))$.

$K_l(\lambda_{kl},x)$ の計算 ... ステップ (4) ($l \approx \sqrt{n}$ 次多項式の $m \approx \sqrt{n}$点での評価)

- m 個の独立な evaluation と扱えば (並列度は m)，各々の Horner による計算量は $O(lM(n))$.
- Chinese remaindering を用いれば，乗算および剰余の各フェイズの各レベルは，$K_l(Y,x)$ の計算での各レベルと同等である [7].

本稿では，Horner を用いる方法を示す．

13.6 Parallel DDF algorithm

13.3節で述べた方針に従い，前節で示した方法を用いた並列処理のための DDF アルゴリズムを 図 2 に具体的に示す．図中の特殊な構文は，次のような並列処理を行うことを示す．

- **parallel-forall** では，指標に関して並列実行を行う．
- **do-in-parallel** では，各項目を並列に実行する．
- **&** の直前の構文は，別タスクとして並列に実行する．

表 2 にステップ毎の計算量をまとめる．

- この記述以上に粒度を細かくしなければ，並列度は，$n^{1/2}$ (もしくは，その 2 倍) である．
- ステップ (1) では，繰り返し i における $O(M(n)\log n)$-time の GCD 計算は，$i+1$ における λ_λ の計算 (冪乗の計算は，高さ (逐次性) が少なくとも $\log n$ だけある) で隠蔽される．
- ステップ (5&6) の GCD 計算は，計算量的には，$k+1$ の coarse DDF と並列実行される k の各 fine DDF とが隠蔽しあう．よって，計算量は高々 $O(n^{1/2}M(n)\log n) \leq O(M(n^{3/2})\log n)$ である．
- 全体の計算量は，$O((M(n^{3/2}) + n^{1/2}MM(n^{1/2}))\log n + M(n)\log q)$ で押えられる．

表 2　並列 DDF アルゴリズム (図 2) のステップ毎の計算量

ステップ	time (既知の上限)	(単純な方法による) 並列度	space
(1)	$O(M(n)\log q)$	1	$O(n)$
	$O(n^{1/2}(M(n) + MM(n^{1/2}))\log n)$	$l/2 = O(n^{1/2})$	$O(n^{3/2})$
	(Horner でも $O(nM(n)\log n)$)		
(2)	同上	$m/2 = O(n^{1/2})$	$O(n^{3/2})$
(3)	$\sum_{k=1}^{\lceil \log_2 l \rceil} O(M(2^{k-1}n))$	$l/2 = O(n^{1/2})$	$O(n^{3/2})$
(4)	$O(M(n^{3/2}))$	$m = O(n^{1/2})$	$O(n^{3/2})$
(5 & 6)	$O(n^{1/2}M(n)\log n)$	$l = O(n^{1/2})$	
$O((M(n^{3/2}) + n^{1/2}MM(n^{1/2}))\log n + M(n)\log q)$ で押えられる			

(1) $\lambda_0(x) \Leftarrow x$; $\lambda_1(x) \Leftarrow x^q \bmod F(x)$;; $f_1 \Leftarrow \gcd(\lambda_1(x) - x, F(x))$;; $F(x) \Leftarrow F(x)/f_1$;;

(* compute $\lambda_k(x)$ in a doubling manner as $k = 2$; $3 \sim 4$; $5 \sim 8$; $9 \sim 16$; $17 \sim 32$; ..., and compute & remove f_k from $F(x)$ *)

for $i = 0$ **to** $\lceil \log_2 l \rceil - 1$ **do**

parallel-forall $k = 1, 2, ..., \max(2^i, l - 2^i)$ **do**

$\lambda_{k+2^i}(x) \Leftarrow \lambda_{2^i}(\lambda_k(x))$;;

{ $f_{k+2^i} \Leftarrow \gcd(\lambda_{k+2^i}(x) - x, F(x))$;;

if $f_{k+2^i} \neq 1$ **then** $F(x) \Leftarrow F(x)/f_{k+2^i}$; (* broadcast recommended *) } &

(1') If $f_i \neq 1$ for some i, adjust $F(x)$ and

parallel-forall $i = 1, 2, ..., l$ **do** $\lambda_i(x) \Leftarrow \lambda_i(x) \bmod F(x)$;;

(2&3) **do-in-parallel**

- **for** $i = 0$ **to** $\lceil \log_2 m \rceil - 1$ **do**

parallel-forall $k = 1, 2, ..., \max(2^i, m - 2^i)$ **do**

$\lambda_{(k+2^i)l}(x) \Leftarrow \lambda_{2^i l}(\lambda_{kl}(x))$;

- Let $K_{i,0}(Y, x) = Y - \lambda_i(x)$, $0 \leq {}^\forall i \leq l - 1$, and $= 1$, $l \leq {}^\forall i < 2^{\lceil \log_2 l \rceil}$;;

for $j = 1$ **to** $\lceil \log_2 l \rceil - 1$ **do**

parallel-forall $i \in \{ k \mid 0 \leq k2^j < l \}$ **do**

$K_{i2^j,j}(Y, x) \Leftarrow K_{i2^j,j-1}(Y, x)\, K_{i2^j+2^{j-1},j-1}(Y, x)$;

(4) Letting $K_l(Y, x) \Leftarrow K_{0,\lceil \log_2 l \rceil - 1}(Y, x)$,

parallel-forall $k = 1, 2, ..., m$ **do** $\Delta_k \Leftarrow K_l(\lambda_{kl}(x), x) \bmod F(x)$;;

(5&6) $k \Leftarrow 2$;;

while $2k \leq \deg F$ **do**

$\gcd(\Delta_k, F) \Rightarrow F_k$;; (* coarse DDF *)

{ **parallel-forall** $i = 0, 1, ..., l - 1$ **do** $\gcd(\lambda_{kl}(x) - \lambda_i(x), F_k) \Rightarrow f_{kl-i}$; } &

$F \Leftarrow F/F_k$;; $k \Leftarrow k + 1$;

図 2 並列 DDF アルゴリズム

13.7 さいごに

本稿では，$O(\sqrt{n})$ 台のプロセッサを利用すれば，全体で $O(n^{3/2})$ の space を用いて，漸近的高速アルゴリズム等の活用により高々 $O(n^2 \log^\alpha n)$ 程度の時間で計算可能であることを示した．modular

composition や evaluation 等を詳細に検討すれば，多項式を複数の項毎に分割することにより，かなり細かい粒度の並列性も出現するが，その場合には通信コストまで考慮する必要があるため，現時点では，充分な検討を行っていない．実際の並列機への実装と実験，及び，それに基づく通信コストも含めた解析は，今後の研究課題である．

謝辞

一方の筆者である村尾 裕一は，本研究の実施にあたり，一部，平成5~6年度 文部省科学研究費補助金一般研究 (C)「数式処理とスーパーコンピューティング」(課題番号 05680266) 及び平成7~8年度 同 (C)「数式処理算法のベクトル処理と並列処理及びその分散協調に関する研究」(課題番号 07680337) の補助を受けました．

参考文献

[1] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] BORODIN, A., AND MUNRO, I. *The Computational Complexity of Algebraic and Numeric Problems.* Theory of Computation Series. American Elsevier Publishing Company, Inc., 1975.

[3] BRENT, R. P., AND KUNG, H. T. Fast algorithms for manipulating formal power series. *J. ACM 25*, 4 (1978), 581–595.

[4] BURKE-PERLINE, T. The parallel computation of $f(x)^{\frac{(p-1)}{2}} \bmod h(x)$ using *sugarbush 1.1.* In Hong [10], pp. 74–83.

[5] CANTOR, D. G., AND KALTOFEN, E. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica 28*, 7 (1991), 693–701.

[6] 藤瀬．並列論理型言語処理系 KLIC による PARI の並列化．京都大学数理解析研究所研究集会「数式処理における理論とその応用の研究」(1994年11月)．同研究所講究録 920 (1995年8月発行), 149–160.

[7] FUJISE, T. AND MURAO, H. Parallel Distinct Degree Factorization Algorithm. (in preparation, 1995).

[8] VON ZUR GATHEN, J. The polynomial factorization challenge. *SIGSAM Bulletin 26*, 2 (1992), 22–24.

[9] VON ZUR GATHEN, J., AND SHOUP, V. Computing Frobenius maps and factoring polynomials. *Computational Complexity 2* (1992), 187–224.

[10] HONG, H., Ed. *Proceedings of PASCO '94* (Linz, Austria, Sept. 26–28 1994).

[11] KALTOFEN, E. Polynomial Factorization 1987–1991. In *Proc. Latin '92*, I. Simon Ed., Springer-Verlag LNCS 583 (1992), pp. 294–313.

[12] KALTOFEN, E. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Mathematics of Computation 64*, 210 (1995), 777–806.

[13] KALTOFEN, E., AND LOBO, A. Factoring hig-degree polynomials by the black box Berlekamp algorithm. In *Proceedings of ISSAC '94* (Oxford, England, July 20–22 1994), J. von zur Gathen and M. Giesbrecht, Eds., pp. 90–98.

[14] KALTOFEN, E., AND SHOUP, V. Subquadratic-time factoring of polynomials over finite fields. (manuscript. obtained from `ftp.cs.rpi.edu@~kaltofen/`), Nov. 1994.

[15] MURAO, H., AND FUJISE, T. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. In Hong [10], pp. 304–315.

[16] 村尾. $\mathbf{Z}_p$ 上の多項式の因数分解 — 高速化技法・ベクトル処理・並列処理 —. 京都大学数理解析研究所研究集会「数式処理における理論とその応用の研究」(1994 年 11 月). 同研究所講究録 920 (1995 年 8 月発行), 133–148.

[17] 村尾, 藤瀬. 多項式因数分解へのスーパーコンピュータの応用, 1995 年 6 月. 第 4 回日本数式処理学会大会 (奈良女子大学).

[18] 佐々木建昭. 数式処理. 情報処理叢書 7. 情報処理学会, 1981.

[19] SHOUP, V. Factoring polynomials over finite fields: asymptotic complexity vs. reality. In *Proc. International IMACS Symposium on Symbolic Computation, New Trends and Development* (Lille, France, 1993), pp. 124–129.